Synthesising Executable Gene Regulatory Networks from Single-cell Gene Expression Data

Jasmin Fisher[1,2], Ali Sinan Köksal[3], Nir Piterman[4], and Steven Woodhouse[1]

[1] University of Cambridge, UK
[2] Microsoft Research Cambridge, UK
[3] University of California, Berkeley, USA
[4] University of Leicester, UK

Abstract. Recent experimental advances in biology allow researchers to obtain gene expression profiles at single-cell resolution over hundreds, or even thousands of cells at once. These single-cell measurements provide snapshots of the states of the cells that make up a tissue, instead of the population-level averages provided by conventional high-throughput experiments. This new data therefore provides an exciting opportunity for computational modelling. In this paper we introduce the idea of viewing single-cell gene expression profiles as states of an asynchronous Boolean network, and frame model inference as the problem of reconstructing a Boolean network from its state space. We then give a scalable algorithm to solve this synthesis problem. We apply our technique to both simulated and real data. We first apply our technique to data simulated from a well established model of common myeloid progenitor differentiation. We show that our technique is able to recover the original Boolean network rules. We then apply our technique to a large dataset taken during embryonic development containing thousands of cell measurements. Our technique synthesises matching Boolean networks, and analysis of these models yields new predictions about blood development which our experimental collaborators were able to verify.

1 Introduction

As biological data becomes more accurate and becomes available in larger volumes, researchers are increasingly adopting concepts from computer science to the modelling and analysis of living systems. Formal methods have been successfully applied to gain insights into biological processes and to direct the design of new experiments [3–5, 12]. New single-cell resolution gene expression measurement technology provides an exciting opportunity for modelling biological systems at the cellular level. Single-cell gene expression profiles provide a snapshot of the true states that cells can reach in the real experimental system, a level of detail which has not been available before [15, 18]. A major challenge for researchers is to move beyond established methods for the analysis of population data, to new techniques that take advantage of single-cell resolution data [14].

Uncovering and understanding the gene regulatory networks (GRNs) which underlie the behaviour of stem and progenitor cells is a central issue in molecular

cell biology. These GRNs control the self-renewal and differentiation capabilities of the stem cells that maintain adult tissues, and become perturbed in diseases such as cancer. They also specify the complex developmental processes that lead to the initial formation of tissues in the embryo. Understanding how to effectively control GRNs can lead to important insights for the programmed generation of clinically-relevant cell types important for regenerative medicine, as well as into the design of molecular therapies to target cancerous cells.

Biological systems can be modelled at different levels of abstraction. At a molecular level, the biochemical events which occur inside a cell can be captured by stochastic processes, given by chemical master equations [24]. These chemical events are fundamentally stochastic, driven by random fluctuations of molecules present at low concentrations and by Brownian motion. Asynchronous Boolean networks abstract away details of transcription, translation and molecular binding reactions and represent the status of each modelled substance as either active (on) or inactive (off), while using non-determinism to capture different options that arise from stochastic behaviour [7, 13, 27]. In the cell, gene activity is controlled by combinatorial logic in which proteins called transcription factors cooperate to physically bind to a regulatory DNA region of a gene and trigger (or inhibit) its transcription. Target genes may in turn code for transcription factors, forming a complex GRN. Asynchronous Boolean networks are particularly well suited to modelling GRNs because the combinatorial logic regulating gene activity can be expressed as a Boolean function. For example, gene X may be activated by either the presence of gene A or by the presence of both genes B and C. The presence of a repressor D may prevent X from becoming triggered by the presence of these activating genes. When modelling the differentiation of a cell using an asynchronous Boolean network, dynamics proceed by a series of single–gene changes. Mature, differentiated cell types correspond to stable attractor states of the model.

Predictions about the modes of interaction between genes resulting from computational analysis can be tested experimentally through a range of assays. For example, if analysis of a model predicts that gene X is activated by gene A, a ChIP (Chromatin ImmunoPrecipitation) assay can be used to assess whether the protein coded for by A binds to a regulatory region of X. Then, perturbations which prevent the binding of A to this region can be introduced, and the effect that this has on the expression of X can be examined.

State–space analyses of hand–built asynchronous Boolean network models based on literature–derived gene regulatory interactions have been successfully applied to model cell fate decisions, and to reproduce known experimental results (e.g., [2,11,13]). Here we address the problem of automatically constructing such models directly from data. If we think of single-cell gene expression profiles as the state space of an asynchronous Boolean network, can we identify the underlying gene regulatory logic that could have generated this data?

We encode the matching of an asynchronous Boolean network to a state space as a synthesis problem and use constraint (satisfiability) solving techniques for answering the synthesis problem. The synthesised network has to match
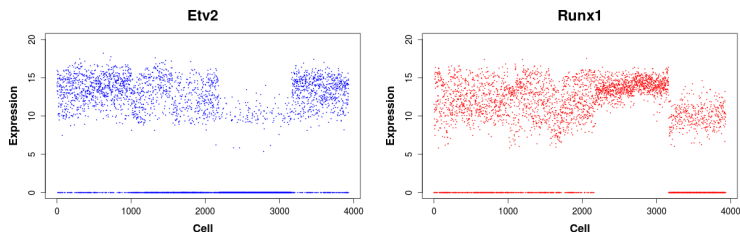
Fig. 1. Single–cell gene expression measurements for two genes, in 3934 cells.

the data in two aspects. First, the resulting network should try to minimise transitions to expression points that are not part of the sampled data. Second, the resulting network should allow for a progression through the state space in a way that matches the flow of time through the different experiments that produced the data. A direct encoding of this problem into a satisfiability problem does not scale well. We suggest a modular search that handles parts of the state space and the network and does not need to reason about the entire network at once. We consider two test cases. First, we try to reconstruct an existing asynchronous Boolean network from its state space. We are able to reconstruct Boolean rules from the original network. Second, we apply our technique to experimental data derived from blood cell development. The network that is produced by our technique matches known dependencies and suggests interesting novel predictions. Some of these predictions were validated by our collaborators.

This paper describes the algorithm that we used to obtain the results in a recently published biological paper on a single-cell resolution study of embryonic blood development [16]. The biological paper includes full details of the experiment that generated the data, and the biological validation of our resulting synthesised model. Here, we cover the algorithmic aspects of our method.

2   Biological Motivation

Single-cell gene expression experiments produce gene expression profiles for individually measured cells. Each of these gene expression profiles is a vector where each element gives the level of expression of one gene in that cell. Figure 1 plots the level of the genes *Etv2* and *Runx1* over 3934 cells.

Our experimental collaborators performed such gene expression profiling on five batches of cells taken from four sequential developmental time points of a mouse embryo. For each time point, the experiment aimed to capture every cell with the potential to develop into a blood cell, providing a comprehensive single–cell resolution picture of the developmental timecourse of blood development. This resulted in a data set of 3934 cell measurements. Full details of this experiment and our analysis can be found in [16]. This data set is the first of its kind, attempting to capture an entire tissue's worth of progenitor cells across a developmental time course. This level of coverage of the potential cell state space is required for our approach to accurately recover gene regulatory networks, and requires the measurement of thousands of cell profiles. Later we will introduce a
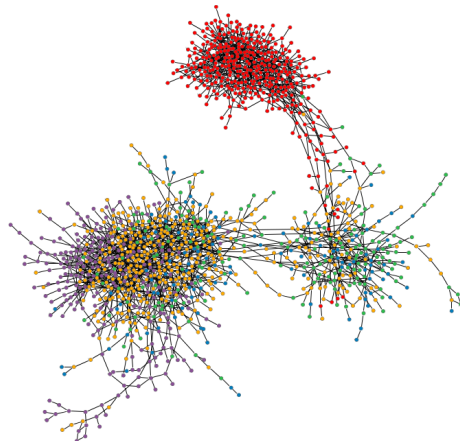
3

Fig. 2. State graph. Node colours correspond to the time point at which a state was measured. States from the earliest of the time points are coloured blue, and states from the last time point are coloured red.

synthetic data set of a few hundred cell states in order to illustrate how our approach works, but we would like to stress that to be usable on real experimental data our algorithm needs to be able to scale thousands of cell states.

For each of 3934 cells, the level of expression of 33 transcription factor genes was measured. Expression levels are non-negative real numbers, where the value 0 indicates that the given gene is unexpressed in the cell (see Figure 1).

The key idea introduced in this paper is to view this gene-expression data as a sample from the state-space of an asynchronous Boolean network. In the past, manually curated Boolean networks have been successfully used to recapitulate experimental results [2, 11, 13]. Such Boolean networks were hand–constructed from biological knowledge that has accumulated in the literature over many years. Here, we aim to produce such Boolean networks automatically, directly from gene expression data, by employing synthesis techniques. We aim to produce a Boolean network that can explain the data and can be used to inform biological experiments for uncovering the nature of gene regulatory networks in real biological systems.

In order to convert the data into a format that can be viewed as a Boolean network state space, we first discretise expression values to binary, assigning the value 1 to all non-zero gene expression measurements. A value of zero corresponds to the discovery threshold of the equipment used to produce the data. Discretising the 3934 expression profiles in this way yields 3070 unique binary states, where every state is a vector of 33 Boolean values corresponding to the activation/inactivation level of each of 33 genes in a given cell. In an asynchronous Boolean network, transitions correspond to the change of value of a single variable. Hence, we next look for pairs of states that differ by only one gene (that is, the Hamming distance between the two vectors is 1). An analysis of the strongly-connected components of this graph shows that one strongly connected component contains 44% of the states. We note that in a random sample

4

of 3934 elements from a space of 2^{33}, the chance of seeing repeats or neighbours with Hamming–distance 1 is negligible.

A plot of the graph of the largest strongly connected component is given in Figure 2. We add an edge for every Hamming–distance 1 pair and cluster together highly connected nodes. The colours of nodes correspond to the developmental time the measurements was taken. Note that there is a clear separation between the earliest developmental time point and the latest one. This representation already suggests a clear change of states over the development of the embryo, with separate clusters identifiable and obvious fate transitions between clusters.

We wish to find an asynchronous Boolean network that matches this graph. For that we impose several restrictions on the Boolean network. Connections between states correspond to a change in the value of one gene, however, we do not know the direction of the change. Thus, we search simultaneously for directions and update functions of the different genes that satisfy the following two conditions: states from the earliest developmental time point should be able to evolve, through a series of single–gene transitions, to the states from the latest developmental time point. Secondly, the update functions must minimise the number of transitions that lead to additional, unobserved states, that were not measured in the experiment.

3 Example: Reconstructing an ABN from its State Space

We first illustrate our synthesis method using an example. We take an existing Boolean network, construct its associated state space, and then use this state space as input to our synthesis method in order to try to reconstruct the Boolean network that we started with.

Krumsiek *et. al.* introduce a Boolean network model of the core regulatory network active in common myeloid progenitor cells [13]. Their network is based upon

Gene	Update function
Gata2	$Gata2 \wedge \neg (Pu.1 \vee (Gata1 \wedge Fog1))$
Gata1	$(Gata1 \vee Gata2 \vee Fli1) \wedge \neg Pu.1$
Fog1	$Gata1$
EKLF	$Gata1 \wedge \neg Fli1$
Fli1	$Gata1 \wedge \neg EKLF$
Scl	$Gata1 \wedge \neg Pu.1$
Cebpa	$Cebpa \wedge \neg (Scl \vee (Fog1 \wedge Gata1))$
Pu.1	$(Cebpa \vee Pu.1) \wedge \neg (Gata1 \vee Gata2)$
cJun	$Pu.1 \wedge \neg Gfi1$
EgrNab	$(Pu.1 \wedge cJun) \wedge \neg Gfi1$
Gfi1	$Cebpa \wedge \neg EgrNab$

Fig. 3. Boolean update functions for a manually curated network.

a comprehensive literature survey. It includes a set of 11 Boolean variables (corresponding to genes) and a Boolean update function for each variable (Figure 3).[5] The model is given a well-defined initial starting state, representing the expression profile of the common myeloid progenitor, and computational analysis reveals an acyclic, hierarchical state space of 214 states with four stable state attractors (Figure 4).

These stable attractors are in agreement with experimental expression profiles of megakaryocytes, erythrocytes, granulocytes and monocytes; four of the mature myeloid cell types that develop from common myeloid progenitors.

We treat the state space of this Boolean network as we would treat experimental data, forgetting all directionality information, and connecting all states

---

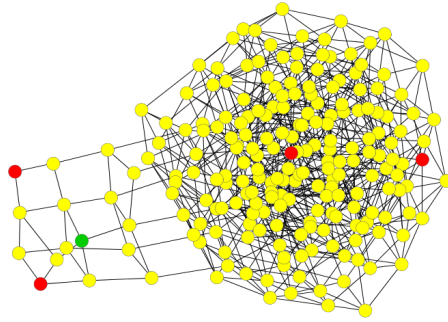[5] The function of *Cebpa* is modified from that in [13] to match the format we assume.

Fig. 4. Boolean network state space. Initial state is coloured green, stable states red.
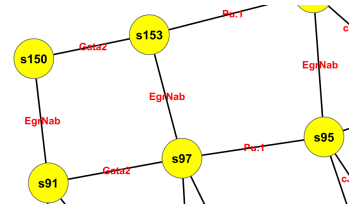

Fig. 5. Close–up of Boolean network state space.

which differ in the expression of only one gene by an undirected edge (Figures 4 and 5, where each edge is labelled with the single gene that changes in value between the states it connects). We would now like to reconstruct the Boolean network given in Figure 3 from this undirected state space.

For each gene, we would like to assign a direction to each of its labelled edges (or decide that it does not exist), in a way that is compatible with a Boolean update function. For example, in Figure 5, we may orient the *Pu.1*-labelled edge between states 97 and 95 in the direction $s_{97} \to s_{95}$, in the direction $s_{95} \to s_{97}$, or decide that this is not a possible update. We also allow the edge to be directed in both directions. If $s_{97} \to s_{95}$, we want a Boolean update function $u_{Pu.1}$ that takes state s_{97} to state s_{95}. Since there is no *Pu.1*–labelled edge leaving state s_{150}, we can also add the constraint that $u_{Pu.1}$ takes s_{150} to s_{150}.

We also add reachability constraints that restrict which edges are included and their orientation. Since the state space was constructed starting from a well-defined initial state, we would like to enforce the constraint that each non-initial state ought to be reachable by some directed path from the initial state. Since cell development proceeds hierarchically and unidirectionally, we favour short paths over long paths. This eliminates routes that seem biologically implausible, for example routes that cross a fate transition and then return to where they began. It also reduces the space of paths we have to search through. By increasing the lengths of allowed paths, we can increase the number of considered solutions.

The results of applying our technique are shown in Figure 6. The method reconstructs the Boolean update functions for all but one gene (*EgrNab*), in some cases uniquely identifying the original function. We note that when multiple solutions are found for an update function, these solutions, while not exact, all provide useful regulatory information that could be verified experimentally. For example, both solutions for *Scl* successfully predict *Scl*'s activation by *Gata1*, although one of the two solutions omits its repression by *Pu.1*.

Gene	Synthesised update functions	Comments
Gata2	$Gata2 \wedge \neg(Fog1 \vee Pu.1)$	
	$Gata2 \wedge \neg(Fog1 \vee (Pu.1 \wedge Cebpa))$	
	$Gata2 \wedge \neg(Fog1 \vee (Pu.1 \wedge Gata2))$	
	$Gata2 \wedge \neg(Gata2 \wedge (Pu.1 \vee Fog1)$	
	$\mathbf{Gata2 \wedge \neg(Pu.1 \vee (Gata1 \wedge Fog1))}$	
	$Gata2 \wedge \neg(Pu.1 \vee (Gata2 \wedge Fog1))$	
Gata1	$(Gata1 \vee Cebpa) \wedge \neg Pu.1$	
	$(Gata2 \vee Fog1) \wedge \neg Pu.1$	
	$(Gata1 \vee Gata2) \wedge \neg Pu.1$	
	$\mathbf{(Gata1 \vee Gata2 \vee Fli1) \wedge \neg Pu.1}$	
	Other functions of the form $(X \vee Y \vee Z) \wedge \neg Pu.1$	
Fog1	$\mathbf{Gata1}$	Unique
EKLF	$\mathbf{Gata1 \wedge \neg Fli1}$	Unique
Fli1	$\mathbf{Gata1 \wedge \neg EKLF}$	Unique
Scl	$Gata1$	
	$\mathbf{Gata1 \wedge \neg Pu.1}$	
Cebpa	$Cebpa \wedge \neg(Fog1 \vee Scl)$	
	$Cebpa \wedge \neg(Cebpa \wedge (Scl \vee Fog1))$	
	$Cebpa \wedge \neg(Fog1 \wedge (Scl \vee Cebpa))$	
	$Cebpa \wedge \neg(Fog1 \vee (Scl \wedge Gata1))$	
	$Cebpa \wedge \neg(Fog1 \vee (Scl \wedge Gata2))$	
	$Cebpa \wedge \neg(Gata1 \wedge (Fog1 \vee Scl)$	
	$Cebpa \wedge \neg(Scl \vee (Fog1 \wedge Cebpa)$	
	$\mathbf{Cebpa \wedge \neg(Scl \vee (Fog1 \wedge Gata1)}$	
Pu.1	$Pu.1 \wedge \neg Gata2$	
	$(Pu.1 \wedge Cebpa) \wedge \neg Gata2$	
	$Pu.1 \wedge \neg(Gata1 \vee Gata2)$	
	Other functions of the form $Pu.1 \wedge \neg(Gata2 \vee X)$	
	$Pu.1 \wedge \neg(Gata2 \wedge Cepba)$	
	$Pu.1 \wedge \neg(Gata2 \wedge Pu.1)$	
	$Cebpa \wedge \neg(Gata1 \vee Gata2)$	
	$Cebpa \wedge \neg(Gata2 \vee Fog1)$	
	$\mathbf{(Cebpa \vee Pu.1) \wedge \neg(Gata1 \vee Gata2)}$	
	$(Cebpa \wedge Pu.1) \wedge \neg(Gata1 \vee Gata2)$	
	Other functions of the form $(Cebpa \vee X) \wedge \neg(Gata2 \vee Y)$	
	Other functions of the form $(Pu.1 \vee X) \wedge \neg(Gata2 \vee Y)$	
	Other functions of the form $(Cebpa \wedge Pu.1) \wedge \neg(Gata2 \vee X)$	
cJun	$\mathbf{Pu.1 \wedge \neg Gfi1}$	Unique
EgrNab	$(cJun \vee Gata1) \wedge \neg Gfi1$	Incorrect with shortest paths
Gfi1	$\mathbf{Cebpa \wedge \neg EgrNab}$	Unique

Fig. 6. Synthesised update functions.

4 Background to Asynchronous Boolean Networks

An *asynchronous Boolean network* (ABN) is $B(V, U)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of *variables*, and $U = \{u_1, u_2, \ldots, u_n\}$ is a set of Boolean *update functions*. For every $u_i \in U$ we have $u_i : \{0,1\}^n \to \{0,1\}$ associated with variable v_i. A *state* of the system is a map $s : V \to \{0,1\}$. We say that an update function u_i is *enabled* at state s if $u_i(s) \neq s(v_i)$, i.e. applying the update function u_i to state s changes the value of variable v_i.

State $s' = (d'_1, d'_2, \ldots, d'_n)$ is a *successor* of state $s = (d_1, d_2, \ldots, d_i, \ldots, d_n)$ if for some i we have u_i is enabled, $d'_i = u_i(s)$, and for all $j \neq i$ we have $d'_j = d_j$. That is, we get to the next state s', by non-deterministically selecting an enabled update function u_i and updating the value of the associated variable: $s' = (d_1, d_2, \ldots, u_i(d_i), \ldots, d_n)$. If no update function is enabled, the system remains in its current, stable, state, where it will remain: $s' = s$.

An ABN induces a labelled transition system $T = (N, R)$, where N is the set of 2^n states of the ABN, and $R \subseteq N \times V \times N$ is the successor relation. Each transition (s_1, v_i, s_2) is labelled with the variable v_i such that $s_1(v_i) \neq s_2(v_i)$.

The *undirected state space* of an ABN is an undirected graph $S = (N, E)$, where each vertex $n \in N$ is uniquely labelled with a state s of the Boolean network, and there is an edge $\{s_1, s_2\} \in E$ iff s_1 and s_2 differ in the value of exactly one variable, v. The edge $\{s_1, s_2\}$ is labelled with v. In general, an undirected state space does not have to include all 2^n states induced by a Boolean network.

An ABN $B(V, U)$ *induces* a *directed state space* on an undirected state space $S = (N, E)$. Consider the transition system $T = (2^V, R)$ of $B(U, V)$. Then, the induced directed state space is $S' = (N, A)$, where $(s_1, s_2) \in A$ implies that there is a variable v_i such that $(s_1, v_i, s_2) \in R$. We say that (s_1, s_2) is *compatible* with u_i, if $s_2(v_i) = u_s(s_1)$, and for every $j \neq i$ we have $s_2(v_j) = s_1(v_j)$.

5 Formal Definition of the Problem

Our synthesis problem can be stated as follows: we are given an undirected state space S over a given set of variables V. We would like to extract a set of Boolean update functions that induce a directed state space from S such that each of the states in S are reachable from a given set of initial states. We also want to ensure that no additional, undesired states not in S are reachable, by ruling out transitions which 'exit' the state space.

More formally, we are given a set of variables $V = \{v_1, v_2, \ldots, v_n\}$, an undirected state space $S = (N, E)$ over V, and a set $I \subseteq N$ of *initial* vertices.

We would like to find an update function $u_i : \{0, 1\}^n \to \{0, 1\}$ for each variable $v_i \in V$, such that the following conditions hold. Let $U = \{u_i \mid v_i \in V\}$ be the set of update functions.

1. Every non-initial vertex $s \in N - I$ is reachable from some initial vertex $s_i \in I$ by a directed path in the directed state space induced by $B(V, U)$ on S.
2. For every variable $v_i \in V$, let N_i be the set of states without an outgoing v_i-labelled arc. For every i we require that for each $s \in N_i$, $u_i(s) = s(v_i)$.

5.1 Generalising the Definition to Partial Data

Since we intend to apply our method in an experimental setting, where we only have an incomplete sample from the possible states of the system, we relax this definition to extend it to partial data. Instead of requiring that *every* state is reachable from those initial states that we have measured, we only require that a set of *final* states are reachable. Instead of requiring that every undesired transition is ruled out, we seek to maximise the number of such transitions which are eliminated. This is formally stated next.

As before, we are given a set of variables $V = \{v_1, v_2, \ldots, v_n\}$, an undirected state space $S = (N, E)$ over V, and a designated set $I \subseteq N$ of *initial* vertices. In addition, we are given a designated set $F \subseteq N$ of *final* vertices, along with a *threshold* t_i for each variable $v_i \in V$. The threshold t_i specifies how many undesired transitions must be ruled out.

We would like to find an update function $u_i : \{0,1\}^n \to \{0,1\}$ for each variable $v_i \in V$, such that the following conditions hold. Let $U = \{u_i \mid v_i \in V\}$ be the set of update functions.

1. Every final vertex $s_f \in F$ is reachable from some initial vertex $s_i \in I$ by a directed path in the directed state space induced by $B(V, U)$ on S.

2. For every variable $v_i \in V$, let N_i be the set of states without an outgoing v_i-labelled arc. For every i the number of states $s \in N_i$ such that $u_i(s) = s(v_i)$ is greater or equal to t_i.

In the remainder of the text, we refer to condition 1 as the *reachability condition* and condition 2 as the *threshold condition*.

We restrict the search to update functions of the form $f_1 \wedge \neg f_2$, where f_i is a monotone Boolean formula. The inputs to f_1 are the activating inputs to the gene and the inputs to f_2 are the the repressing inputs. This restriction was chosen after discussion with biologist colleagues and consultation of the literature (e.g., [2, 13]).

6 A Direct Encoding

We start with a direct encoding of the search for a matching Boolean network. The search is parameterised by the shape of update functions (how many activators and how many repressors each variable has), the length of paths from initial states to final states, and the thresholds for each variable. By increasing the first two parameters and decreasing the last we can explore all possible Boolean networks.

6.1 Possible Update Functions

In order to represent the Boolean update function for gene v_i, $u_i = f_1 \wedge \neg f_2$, we use a bitvector encoding. We represent the Boolean formula f_j by a set of bitvectors, $\{a_1, a_2, \ldots a_n\}$, $a_j \in V \cup \{\vee, \wedge\}$, where each bitvector a_i represents a variable or a Boolean operator, and solutions take the form of a binary tree. For example, the formula $v_1 \wedge (v_2 \vee v_3)$ is represented by the solution $a_1 = \wedge, a_2 = \vee, a_3 = v_1, a_4 = v_2, a_5 = v_3$. We restrict the syntactic form of possible update functions so that each variable appears only once, and each possible function has one canonical representation. For example, the function $(v_1 \wedge (v_2 \vee v_3))$ is included in our search space while $(v_1 \wedge v_2) \vee (v_1 \wedge v_3)$ is not. We search for functions up to a maximum number of activators, A_i, and a maximum number of repressors, R_i.

To encode the application of function u_i to a state s, $u_i(s)$, we add implications which unwrap the bitvector encoding of u_i to the constituent variables and logical operators; substituting values, $s(v_j)$, for variables, v_j, and directly mapping operations to logical constraints in the Boolean satisfiability formula. For example, the application of the function $(v_1 \vee v_2) \wedge \neg v_3$ to the state s_1 is mapped to $(s_1(v_1) \vee s_1(v_2)) \wedge \neg s_1(v_3)$.

6.2 Ensuring Reachability

To enforce the global reachability condition we consider all of the underlying directed edges in the undirected state space $S = (N, E)$, and their associated single–gene transitions.

Recall that we require every final vertex to be reachable from some initial vertex by a directed path in the directed state space induced on S by the Boolean network. That is, we require that every final vertex is reachable by a directed path, and that every v_j-labelled edge along this path is compatible with its associated update function, u_j.

To enforce this we add constraints that track the compatibility of edges with update functions and define reachability recursively. We consider reachability by paths up to a maximum length: recall that we consider shorter paths to be more biologically likely. By iteratively increasing the length of the paths considered, we can obtain all satisfying models.

We introduce a pair of Boolean variables e_{ij}, e_{ji} for each v_i-labelled undirected edge $\{s_i, s_j\} \in E$, which track the value of the application of u_i to s_i and to s_j (and the compatibility of the underlying directed edges (s_i, s_j) and (s_j, s_i) with u_i). e_{ij} is true iff $u_i(s_i) = s_j(v)$.

We introduce an integer given by a bitvector encoding, r_n, for each node $n \in N$. Bitvector r_n encodes the fact that node n is reachable from an initial node in r_n steps, up to some maximum encodable value $2^{|r_n|} - 1$. Bitvector r_n is given a value of -1 to indicate that n is not reachable in this maximum number of steps.

Reachability is then defined inductively:

1. Initial nodes are reachable in zero steps: for every $i \in I$, $r_i = 0$.
2. A non–initial node s_i is reachable in M steps if there is a compatible incoming edge (s_j, s_i) from another node s_j, and s_j is itself reachable in fewer than M steps. That is, for every $n = s_j \in N - I$ and $m = s_i \in N$ such that $\{s_i, s_j\} \in E$ we have $e_{ij} \to r_m < r_n$. We also have that non–initial nodes cannot be reached in zero steps: For every $n \in N - I$, $r_n = -1 \vee r_n > 0$.

Finally, we add a constraint that every final node $n \in F$ is reachable from some initial node: $r_n \neq -1$.

6.3 Enforcing the Threshold Condition

We enforce the threshold condition for each update function as follows.

Consider an update function $u_i : V \to \{0, 1\}$. We say that a node $s \in N_i$ is *negatively matched* by u_i if $u_i(s) = s(v_i)$. That is, by using u_i as the update function of variable v_i, u_i does not change the value of v_i from node s. We are searching for an update function such that a maximum number of nodes from N_i are negatively matched.

We add a variable, m_{is} for each node $s \in N_i$ to record whether u_i negatively matches s. We then add a constraint demanding that the number of negatively matched nodes is greater than or equal to the threshold: $\sum_{s \in N_i} m_{is} \geq t_i$.

We search for satisfying assignments to the constraint variables encoding the representation of the Boolean update functions u_i for all v_i in V. The resulting synthesised Boolean network is the combination of these update functions.

Unfortunately, in practice the direct encoding of the search does not scale to handle our experimental data. In the next section we suggest a compositional way to solve the problem.

7   A Compositional Algorithm

We now introduce our compositional algorithm, which scales better than the direct encoding given above. The problem of synthesising a Boolean network from the data is partitioned to three stages. Crucially, we avoid searching for a complete Boolean network and consider parts of the network that can be constructed independently.

7.1   Pruning the Set of Possible Edges

We start by building a directed graph from the given undirected state space $S = (N, E)$, by considering which of the underlying directed edges in E are compatible with some Boolean update function, and pruning those that are not. We consider each underlying directed edge (s_1, s_2) and (s_2, s_1) of each of the v_i-labelled undirected edges $\{s_1, s_2\}$ in E independently.

We pose a decision problem for each directed edge (s_1, s_2): whether there exists some Boolean update function u_i satisfying the threshold condition (condition 2, 5.1) such that $u_i(s_1) = s_2(v_i)$. This is encoded as a Boolean satisfiability problem, adding constraints to represent the encoding of the update function, the threshold condition, and the evaluation of the function at the specific edge under consideration. We say that a satisfying function, u_i, is *compatible* with (s_1, s_2). Once a compatible function has been found, it can quickly be evaluated outside the solver at other edges to try reduce the number of SAT queries we have to make.

After making a query for each edge, we are left with a directed graph, which is the existential projection of all compatible update functions for each of the variables $v \in V$. We have eliminated edges which have no compatible update function, and cannot participate in the reachability condition. On the example data set from Section 3, this step removes 18% of the possible edges.

7.2   Ensuring Reachability

We now come to the only part of the algorithm that considers the edges of all variables together, in order to enforce the global reachability condition (condition 1, 5.1). This phase does not require the solving of a Boolean satisfiability problem, and as a result is very efficient.

We construct, for each pair of initial nodes $i \in I$ and final nodes $f \in F$, the shortest path p_{if} from i to f in the directed graph that was built in the previous phase of the algorithm. These paths can be computed via a breadth–first search.

Due to the edge pruning of the previous phase of the algorithm, if there is no path to a final node f, this implies that there are no satisfying models (at the given threshold and function size parameters). Otherwise, our reachability condition will be enforced by fixing a set of directed edges P_i for each variable $v_i \in V$ corresponding to these shortest paths. We will then require that the update function we search for, u_i, is compatible with each of the edges in P_i.

We choose, for each final node f, one path $p_f = p_{if}$ from one of the initial nodes i. By fixing this path, we ensure that f is reachable from an initial node. We define $p_f|_i$ as the set of v_i-labelled edges in the path p_f. We define P_i, the v_i-labelled edges which must be fixed to ensure reachability via the chosen paths, as the the set of v_i-labelled edges in p_f for each final node f:

$$P_i = \bigcup_{f \in F} \{(s_1, s_2) \mid (s_1, s_2) \in p_f|_i\} \tag{1}$$

By considering only the edges in P_i, we can search for an update function for v_i independently of all other variables, while ensuring the global reachability condition holds.

7.3   Final Update Functions

We can now search for the update function of variable v_i, u_i, independently of all other variables. We fix the v_i-labelled edges computed in the previous phase and encode the search for u_i as a Boolean satisfiability problem.

As before we add constraints to encode the representation of u_i, and to enforce the threshold condition. We fix each of the v_i-labelled edges $(s_1, s_2) \in P_i$ to establish reachability, by adding a conjunction requiring that u_i is compatible with each of them: $u_i(s_1) = s_2(v_i)$.

We search for satisfying assignments of the constraint variables encoding u_i, using an ALLSAT procedure to extract all possible update functions for variable v_i. This gives rise to a set of update functions per variable and a set of Boolean networks from the product of the set of update functions per variable.

We note that this final phase of the algorithm can fail to find update functions for a variable v_i, because there are no possible update functions compatible with all of the path edges P_i that were computed in the previous phase. That is, while each edge in P_i is individually compatible with some update function, there may be no update function that is compatible with every edge in P_i. In order to cope with this limitation, we can extract the minimal unsatisfiable core of the Boolean formula, and search for replacement paths that exclude incompatible combinations of edges. This step can be iterated until satisfying solutions are found for all variables, or until no path can be found, implying that there are no valid models.

By extending our search from the shortest paths between initial and final node pairs in the directed graph to the k-shortest paths between pairs and incremementally increasing k [26], we can increase the number of possible update functions that we consider. In the limit, we will obtain all satisfying models.

Data set	Genes	States	Direct (seconds)	Compositional (seconds)
CMP (synthetic)	11	214	25	77
Blood stem cells	21	753	OUT OF MEMORY	5114
Embryonic (66% of states)	33	956	OUT OF MEMORY	3364
Embryonic (full)	33	1448	OUT OF MEMORY	8709

Fig. 7. Performance of direct encoding and compositional algorithm on example data sets.

An implementation of our algorithm, which is written in F# and uses Z3 as the satisfiability solver, is available at `https://github.com/swoodhouse/SCNS-Toolkit`. In Figure 7 we present experimental results from running our implementation of the direct encoding from Section 6 and compositional algorithm on four data sets: the small synthetic data set from Section 3, the large embryonic experimental data set from Section 2, and a second experimental data set covering blood stem cells. We also show results from rerunning on the embryonic data set with a third of states removed. All experiments were performed on an Intel Core i5 @ 1.70GHz with 8GB of RAM, using a single thread.

While the direct encoding synthesised a matching Boolean network on the small synthetic data set faster than our compositional algorithm, it cannot scale to the real experimental data sets, quickly running out of memory. The compositional algorithm, on the other hand, can scale to handle real data sets of the sort produced by our experimental collaborators. All experiments terminated within a few hours, when running on a single thread. The compositional algorithm can easily be parallelised over variables, which would further increase its efficiency.

8 Application to the Experimental Dataset

We now return to the experimental data set introduced in Section 2.

Recall that cell measurements were taken from four sequential developmental time points, and that the state graph resulting from discretisation of the data (Figure 2) exhibited a clear separation between the earliest developmental time point (states coloured blue) and the latest (states coloured red). We applied our synthesis technique to this data, taking the initial states to be the states from the first time point, and the final states to be the states from the latest time point. For complete details, we direct the reader to [16].

The result of the synthesis was a set of possible Boolean update functions for each of the 33 genes, with several genes having a uniquely identified update function. By applying standard techniques for the analysis of Boolean networks, we found the stable state attractors and performed computational perturbations. The synthesised network, along with the subsequent computational analysis led to a set of predictions which were then tested experimentally. We found that our results were robust when performing bootstrapping, removing a third of the data at random and rerunning the synthesis algorithm.

Our experimental collaborators were able to validate key predictions made by our analysis. The update function for one of the genes at the core of this network, *Erg*, which directly activates many other genes, was tested experimentally by a

range of assays. Evidence was found that the activators specified in the gene's synthesised update function (*Hoxb4* and *Sox17*) do indeed activate expression of the gene, and furthermore in a fashion consistent with the Boolean "OR" logic of the synthesised update function. This could be regarded as a "local" validation of our model, testing two of the directed edges in the network.

Computational perturbations to another gene at the core of the network, *Sox7*, indicated that when *Sox7* was forced to always be expressed, stable states corresponding to cells from the final developmental time point (blood progenitors) no longer exist. Cell differentiation assays confirmed this prediction experimentally, finding that when this gene was forced to be expressed, the number of cells which normally emerge at this final time point is significantly reduced. This can be thought of as a "global" validation of our model, as it is a prediction about the behaviour of the whole network under a certain perturbation.

9   Related Work

Previous analyses of single-cell gene expression data have mostly been based on statistical properties of the data viewed as a whole, such as the correlation in the level of expression of pairs of genes [8, 15]. Such analysis cannot recover mechanistic Boolean logic, does not infer the direction of interactions and cannot easily distinguish direct from indirect influence.

Boolean networks were introduced by Kauffman in order to study random models of genetic regulatory networks [10]. They have since been applied in a range of contexts, from modelling blood stem and progenitor differentiation [2,13], to the yeast apoptosis network [11], to the network regulating pluripotency in embryonic stem cells [9]. BDD-based algorithms for state-space exploration and finding attractors of Boolean networks have been introduced [7,27].

Synthesis is the problem of producing programs or designs from their specifications. In recent years much progress has been made on the usage of SAT and SMT solvers for synthesis. Essentially, the existence of a program that solves a certain problem is posed as a satisfiability query. Then, a solver tries to search for a solution to the query, which corresponds to a program. For example, Srivastava *et. al.* [22,23] show that the capabilities of SMT solvers to solve quantified queries enable the search for conditions and code fragments that match a given specification. Similarly, Solar-Lezama *et. al.* [21] build a framework for writing programs with "holes" and letting a search algorithm find proper implementations for them. The approach of reactive synthesis [19] is similar to ours in the type of artefact that it produces. However, the techniques that we are using are more related to those explained above. Recently, Beyene *et. al.* [1] have shown how constraint solving can be used also in the context of reactive synthesis.

Synthesis has recently been applied in the context of biology. Köksal *et. al.* show how to synthesise state-machine-like models from gene mutation experiments using a novel counterexample-guided inductive synthesis (CEGIS) algorithm [12]. Their approach uses constraint solvers to search for program completions that match given specifications, as explained above. Both the data and the type of model are different to those dealt with here, which called for a new approach.

Recently, there have been several applications of synthesis to Boolean networks. Dunn *et. al.* [6] and Xu *et. al.* [25] show how to fit an existing static, topological regulatory network for embryonic stem cells to gene expression data in order to obtain an executable Boolean network, under the assumption that experimentally measured data represent stable states of the system. This assumption may be appropriate for cell lines maintained in culture, but it does not adapt well to developmental processes such as ours, where cells are transiting through intermediate states in order to develop into a particular lineage.

Recent work of Karp and Sharan [20] shows how to synthesise Boolean networks given a topological network and a set of perturbation experiments, by reduction to integer linear programming. In [17], Paoletti *et. al.* synthesise a related class of models (which incorporate timing and spatial information) from perturbation data, via reducion to SMT. To the best of our knowledge, our approach is the first to synthesise gene regulatory network models directly from raw gene expression data, without the need of either genetic perturbation data or *a-priori* information about the topology of the network.

10   Conclusions and Future Work

We presented a technique for synthesising Boolean networks from single–cell resolution gene-expression data. This new and exciting type of data allows us to consider the state of each cell separately, giving rise to "state snapshots", which we treat as the states of an asynchronous Boolean network. Our key insight is that the update functions of each variable can be sought after separately, giving rise to reasonably sized satisfiability queries. We then combine the single gene update functions by considering the flow of time included in the data.

We are able to reconstruct rules from a manually curated Boolean network and produce a set of possible Boolean networks for the given experimental data, for which no similar curated Boolean network is available. The discussion with biologists about this Boolean network led to a set of predictions, which were then experimentally validated in the lab.

We are awaiting similar data from additional experiments to apply the same technique to. At the same time, we are considering the usage of advanced search techniques, as used in this paper, to the analysis of other types of high-throughput data. Future work in the experimental domain includes the validation of more of the links in our synthesised network, and the design of further gene perturbation experiments motivated by the results of computational perturbations. An interesting question for future research is whether techniques like ours, which achieve scalability by treating different aspects of a graph data structure seperately, are applicable to other domains where graph–like data is generated.

15

References

1. T. A. Beyene, S. Chaudhuri, C. Popeea, and A. Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *41st Symposium on Principles of Programming Languages*, pages 221–234. ACM, 2014.

2. N. Bonzanni, A. Garg, K. A. Feenstra, J. Schtte, S. Kinston, D. Miranda-Saavedra, J. Heringa, I. Xenarios, and B. Gottgens. Hard-wired heterogeneity in blood stem cells revealed using a dynamic regulatory network model. *Bioinformatics*, 29(13), 2013.

3. K. Claessen, J. Fisher, S. Ishtiaq, N. Piterman, and W. Qinsi. Model-checking signal transduction networks through decreasing reachbility sets. In *25th Conference on Comptuer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 85–100. Springer-Verlag, 2013.

4. B. Cook, J. Fisher, B. A. Hall, S. Ishtiaq, G. Juniwal, and N. Piterman. Finding instability in biological models. In *Twenty Sixth International Conference on Computer Aided Verification*, 2014.

5. B. Cook, J. Fisher, E. Krepska, and N. Piterman. Proving stabilization of biological systems. In *Verification, Model Checking, and Abstract Interpretation*, volume 6538 of *Lecture Notes in Computer Science*, pages 134–149. Springer, 2011.

6. S.-J. Dunn, G. Martello, B. Yordanov, S. Emmott, and A. G. Smith. Defining an essential transcription factor program for nave pluripotency. *Science*, 344(6188):1156–1160, 2014.

7. A. Garg, A. Di Cara, I. Xenarios, L. Mendoza, and G. De Micheli. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917–1925, 2008.

8. G. Guo, S. Luc, E. Marco, T.-W. Lin, C. Peng, M. A. Kerenyi, S. Beyaz, W. Kim, J. Xu, P. P. Das, T. Neff, K. Zou, G.-C. Yuan, and S. H. Orkin. Mapping cellular hierarchy by single-cell analysis of the cell surface repertoire. *Cell Stem Cell*, 13(4):492–505, 2013.

9. P. H., R. Abu Dawud, A. Garg, Y. Wang, J. Vilo, I. Xenarios, and A. J. Qualitative modeling identifies IL-11 as a novel regulator in maintaining self-renewal in human pluripotent stem cells. *Front Physiol*, 2013.

10. S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1969.

11. L. Kazemzadeh, M. Cvijovic, and D. Petranovic. Boolean model of yeast apoptosis as a tool to study yeast and human apoptotic regulations. *Front Physiol*, 3, 2012.

12. A. Koksal, Y. Pu, S. Srivastava, R. Bodik, N. Piterman, and J. Fisher. Synthesis of biological models from mutation experiments. In *POPL*, 2013.

13. J. Krumsiek, C. Marr, T. Schroeder, and F. J. Theis. Hierarchical differentiation of myeloid progenitors is encoded in the transcription factor network. *PLoS One*, 6(8), 2011.

14. V. Moignard and B. Gottgens. Transcriptional mechanisms of cell fate decisions revealed by single cell expression profiling. *Bioessays*, 2014.

15. V. Moignard, I. Macaulay, G. Swiers, F. Buettner, J. Schutte, F. Calero-Nieto, S. Kinston, A. Joshi, R. Hannah, F. Theis, S. Jacobsen, M. de Bruijn, and B. Gottgens. Characterization of transcriptional networks in blood stem and progenitor cells using high-throughput single-cell gene expression analysis. *Nat Cell Biol*, 15(4):363–72, 2013.

16. V. Moignard, S. Woodhouse, L. Haghverdi, J. Lilly, Y. Tanaka, A. Wilkinson, F. Buettner, I. Macaulay, W. Jawaid, E. Diamanti, S. Nishikawa, N. Piterman,

V. Kouskoff, F. Theis, J. Fisher, and B. Gottgens. Decoding the regulatory network of early blood development from single-cell gene expression measurements. *Nature Biotechnology*, 2015.

17. N. Paoletti, B. Yordanov, Y. Hamadi, C. M. Wintersteiger, and H. Kugler. Analyzing and synthesizing genomic logic functions. In *CAV'14*. Springer, July 2014.

18. C. Pina, C. Fugazza, A. J. Tipping, J. Brown, S. Soneji, J. Teles, C. Peterson, and T. Enver. Inferring rules of lineage commitment in haematopoiesis. *Nat Cell Biol*, 14:28794, 2012.

19. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th Symposium on Principles of Programming Languages*, pages 179–190. ACM Press, 1989.

20. R. Sharan and R. M. Karp. Reconstructing boolean models of signaling. *Journal of Computational Biology*, 20(3):249–257, 2013.

21. A. Solar-Lezama, R. M. Rabbah, R. Bodík, and K. Ebcioglu. Programming by sketching for bit-streaming programs. In *Programming Language Design and Implementation*, pages 281–294. ACM, 2005.

22. S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In *37th Symposium on Principles of Programming Languages*, pages 313–326. ACM, 2010.

23. S. Srivastava, S. Gulwani, and J. S. Foster. Template-based program verification and program synthesis. *Software Tools for Technology Transfer*, 15(5-6):497–518, 2013.

24. D. Wilkinson. *Stochastic Modelling for Systems Biology*. Chapman and Hall CRC, 2 edition, 2012.

25. H. Xu, Y.-S. Ang, A. Sevilla, I. R. Lemischka, and A. Ma'ayan. Construction and validation of a regulatory network for pluripotency and self-renewal of mouse embryonic stem cells. *PLoS computational biology*, 10(8):e1003777, 2014.

26. J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–16, 1971.

27. D. Zheng, G. Yang, X. Li, Z. Wang, F. Liu, and L. He. An efficient algorithm for computing attractors of synchronous and asynchronous boolean networks. *PLOS ONE*, 2013.